POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

# Systems and Methods for Big and Unstructured Data Project

Author(s): **Gabriele Ginestroni**
**Giacomo Gumiero**
**Lorenzo Iovine**
**Nicola Landini**
**Francesco Leone**

Group Number: **10**

Academic Year: 2022-2023

# Contents

# 1 | Introduction

In this chapter will be presented the problem specification and the hypothesis under which the database is implemented.

## 1.1. Problem Specification

This project aims to build an Information System that handles scientific articles contained in the DBLP bibliography. The project involves managing the type of the articles and the associated DOI (Digital Object Identifier), which identifies a publication or a document and links to it on the web. Other entities to deal with are authors, identified by an ID or ORCID (Open Researcher and Contributor ID), and their affiliations with organizations. In order to address the problem, we will store data in a graph database, allowing us to visualize relations and handle information correctly.

## 1.2. Assumptions

1. All the data in the dataset are heterogeneous, so fields are different
2. The **authors** with missing field *_id* are not considered
3. It is possible that an author writes for different organizations
4. Field *_id* in **author** is unique
5. Field *_id* in **article** exists and it is unique
6. It is impossible that 2 different articles are on the same journal, in the same *volume* with an intersection between *page_start* and *page_end*
7. The designed model doesn't take into consideration the URL associated to the article node, as the main focus of the project was not reading the article
8. It is possible to find a self-reference in a publication
9. A *venue* can be instantiated as a journal, a conference or a generic venue!!!!!!!!

# 2 | ER Diagram

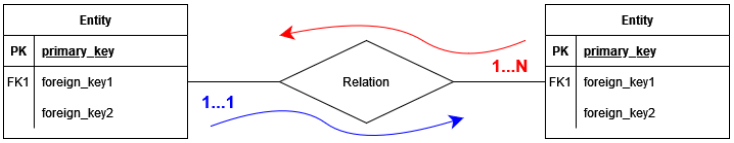


Figure 2.1: ER Diagram Organization

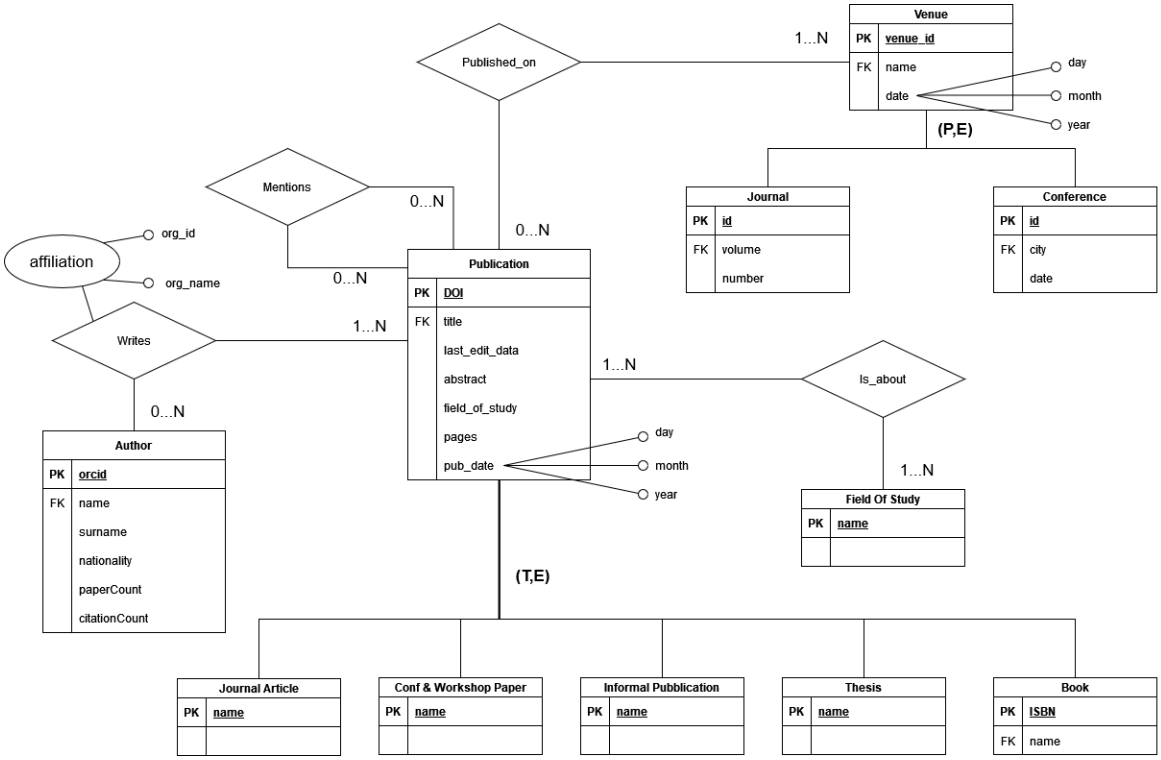


Figure 2.2: ER Diagram

The ER diagram designed contains the following entities:

- **Publication:** this entity represents all the scientific articles. They are identified by their primary key *_id* and other important attributes are: *DOI, title, last_edit_data, abstract, keywords, pages, pub_date*. Of course the attributes of such entity could be enlarged, but as a sample dataset we have believed these are enough. Publication entity is the superclass of a Total and Exclusive ISA relationship with the following subclasses: *Journal article, Conference & Workshop paper, Informal publication, Thesis, Book*
- **Author:** it represents all the people that submitted at least one publication. Its primary key is *_id* and the foreign keys are: *name, surname, nationality, paperCount, citationCount*. Of course the attributes of such entity could be enlarged, but as a sample dataset we have believed these are enough
- **Venue:** it's the entity that represents the type of a publication. This is a superclass that creates a Partial and Exclusive ISA relationship with the two subclasses *Journal* and *Conference*. The primary key is *raw* and the other keys are: *name, date, venue_id*
- **Field Of Study:** this entity represents the topics of the related publication

The ER diagram designed contains the following relationships:

- **Writes:** is the relationship between *author* and *publication* which specifies also the affiliation with the org_id and the org_name
- **Mentions:** occurs between two *publication* and specifies when a publication refers to another one
- **Is about:** binds a *publication* with its *fields of study*
- **Published on:** simply relates a *publication* to its *venue*

# 3 | Dataset Description

## 3.1. Dataset Preprocessing

The dataset we used is based on DBLP-Citation-network V13 at `https://www.aminer.org/citation` whose size is 13GB. Using **Pandas Profiling** we analyzed the dataset focusing our attention on distinctness and completeness of the attributes. We used this result for selecting primary keys (attributes with high values of distinctness and low missing values), to ignore fields that were not informative, and also to filter the tuples. Then we sliced the entire dataset obtaining a subset of 16MB. The slicing operation was not performed randomly in order not to obtain a subset of the dataset containing some publications but any of the publications referenced by them. The working sample of 6400 publications was obtained in this way:

- first a partition of 20M rows was extracted from the original database so that we started from a pool of 257K articles instead of using all the original database of 5.4M articles, we used 5% as a sample to speed up the sample generation process
- then an extract of 4K publications was used as a base, we arbitrarily picked the ones from position 20000 to 24000
- in the end the ids of those 4K publications were looked up inside our partition to obtain also articles that cited them, in order to have sufficient relationships to make different queries

The following is a part of the script used for the operation just described:

```
sed -n $start","$end"p" $input > tmp.json
while IFS='' read -r line || [[ -n "$line" ]]; do
  artId=$(echo "$line" | cut -d '"' -f4)
  grep "$artId" $input >> $output
done < tmp.json
```

In the end, we have to say that we didn't add extra data to our slice of the dataset in order to maintain the coherency of the information.

## 3.2. Attributes Description

In this section we will present all the attributes contained, pointing which of them are considered or not.

### 3.2.1. Publication

Publication represent the central concept of the system and contains:

- **_id** is an alphanumeric string that is the primary key, that's because is unique and every nodes owns this parameter
- **title** represents the title of the publication
- **authors** is an array of authors that will be presented later
- **venue** defines an entity that will be presented later
- **year** represents the year of publication
- **keywords** is an array containing the tag of subjects faced in the publication
- **fos** is an array containing the fields of study of the publication
- **n_citation** is the number of times that this publication has been mentioned
- **page_start** defines the starting page of the publication. This attribute wasn't take into consideration because doesn't target the goal of the project
- **page_end** defines the last page of the publication. This attribute wasn't take into consideration because doesn't target the goal of the project
- **lang** represents the language of the publication
- **volume** is the volume of the publication. This attribute is used in the relationship between *Article* and *Venue*
- **issue** refers to how many times a periodical has been published during that year. This attribute wasn't take into consideration due to the presence of many missing or null values
- **issn** is an identification code of the publication. This attribute wasn't take into consideration due to the presence of many missing or null values
- **isbn** is an identification code of the publication. This attribute wasn't take into consideration due to the presence of many missing or null values

- **doi** Digital Object Identifier is a persistent identifier. We decided to take it into consideration due to an acceptable missing percentage, much lower than the one affecting issn or isbn attributes
- **pdf** contains a string that links to the publication PDF online. This attribute wasn't take into consideration due to the presence of many missing or null values
- **url** contains an array of links to the publication resources online. This attribute wasn't take into consideration because doesn't target the goal of the project
- **abstract** is a string containing a brief summary of the contents of the paper
- **references** is an array of ids representing the publication mentioned

### 3.2.2. Author

Author is the most present entity of the system and contains:

- **_id** is an alphanumeric string that is the primary key, that's because is unique and almost every nodes owns this parameter
- **name** is the name of the author
- **org** is a string that represents the organization in which the author works. It is used as an attribute of the relationship *Writes* described before
- **gid** is an identifier that represents the organization in which the author works. This attribute wasn't take into consideration due to the presence of many missing or null values
- **orgid** is an identifier that represents the organization in which the author works. It is used as an attribute of the relationship *Writes* described before
- **orgs** is an array of organizations for which the author worked. This attribute wasn't take into consideration due to the presence of many missing or null values
- **email** is a string containing the email address of the author. This attribute wasn't take into consideration due to the presence of many missing or null values and because doesn't target the goal of the project
- **orcid** Open Researcher and Contributor ID is a unique identifier for authors of scientific articles. This attribute is taken into consideration although is not always present

- **oid** is an identifier for the author. This attribute wasn't take into consideration due to the presence of many missing or null values
- **bio** is a string that describes the author. This attribute wasn't take into consideration due to the presence of many missing or null values
- **sid** is an identifier for the author. This attribute wasn't take into consideration due to the presence of many missing or null values
- **name_zh** is the name of the organization in which the author works. This attribute wasn't take into consideration due to the presence of many missing or null values
- **org_zh** is an identifier that represents the organization in which the author works. This attribute wasn't take into consideration due to the presence of many missing or null values

### 3.2.3. Venue

Venue is the entity that represents the type of a publication. Thanks to the profiling, we found that venue nodes contains the following attributes:

- **_id** is an alphanumeric identifier. This attribute is not used as a primary key due to the large amount of missing values
- **raw** is the name or the abbreviation of the event or volume (without specifying the year) in which the publication was presented. This attribute was chosen as the primary key thanks to the low number of missing values and to the fact that in this way publications were grouped based on the event in which they were presented
- **raw_zh** refers to the event (without specifying the year) in which the publication was presented. This attribute wasn't take into consideration due to the presence of many missing or null values
- **type** indicates the type of the publication. Exploiting the profiling we understood the distribution of different values of *type*, and become clear that 0 and 1 were attributable respectively to conference and journal. The other values were not easily attributable to other types of publication, so we decided to consider as a generic venue:
  - every *type* values different from 0 or 1
  - every entity in which the *type* field is missing

- **sid** name of the journal in which the publication was published. This attribute wasn't take into consideration due to the presence of many missing or null values
- **t** represents the type of the publication. This attribute wasn't take into consideration due to the presence of many missing or null values
- **issn** is an identification code of the publication
- **name_d** is the extended name of the event or volume (without specifying the year) in which the publication was presented
- **publisher** is a string containing the name of the publisher
- **online_issn** is an identification code of the publication

# 4 | Graph Diagram

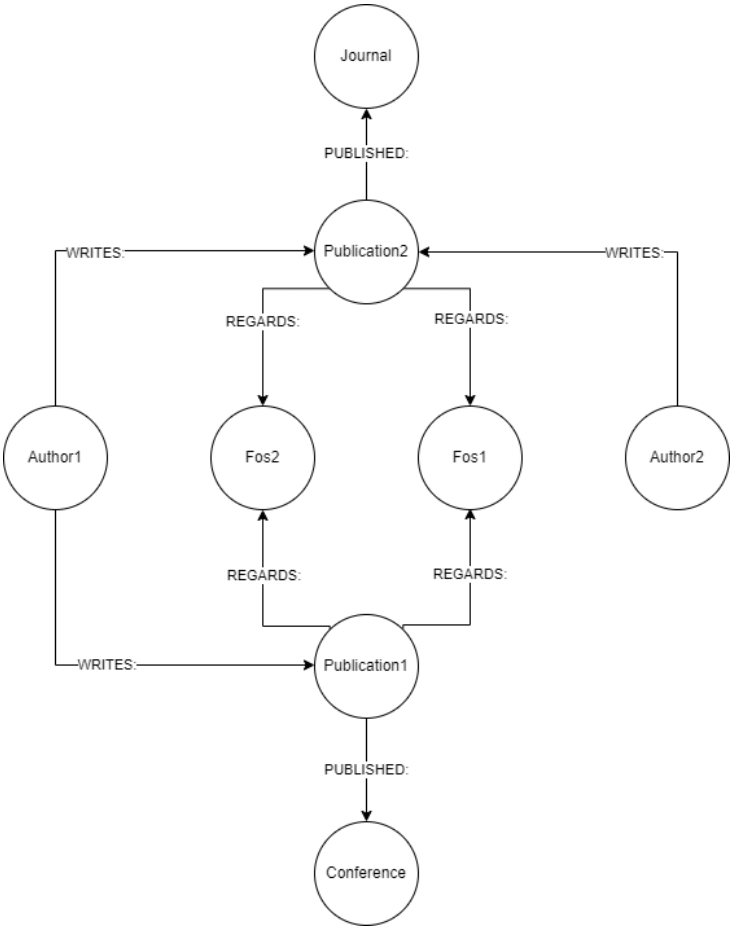


Figure 4.1: Graph Diagram

# 5 | Sample Dataset

In this chapter we will present the import commands that generates 39550 nodes distributed as follows:

- 6340 *Publication* nodes
- 17025 *Author* nodes
- 2133 *Conference* nodes
- 256 *Journal* nodes
- 506 *Generic Value* nodes
- 13210 *Field of Study* nodes

In order to complete these operations we used the plug-in apoc.

1. Create *Publication*, *Author* and `WRITES` relationship between them:

```
call apoc.load.json("test.json") yield value
UNWIND value AS pub
UNWIND pub.authors AS aut WITH aut,pub WHERE aut._id IS NOT NULL
MERGE (publication:Publication{id:pub._id}) ON CREATE SET
    publication.title  = pub.title,
    publication.doi = pub.doi,
    publication.year = pub.year,
    publication.n_citation = pub.n_citation,
    publication.keywords = pub.keywords,
    publication.abstract = pub.abstract,
    publication.lang = pub.lang
MERGE(author:Author{id:aut._id}) ON CREATE SET
    author.name = aut.name,
    author.orcid = aut.orcid
MERGE(author)-[writes:WRITES]->(publication) ON CREATE SET
    writes.org=aut.org,
    writes.orgid=aut.orgid
```

```
```

Added 23365 labels, created 23365 nodes, set 138863 properties, created 18534 relationships, completed after 104197 ms.

2. Create REFERENCE relationship between *Articles*:

```
call apoc.load.json("test.json") yield value
UNWIND value AS pub
UNWIND pub.references AS ref
MATCH (init:Publication{id:pub._id})
MATCH (final:Publication{id:ref})
MERGE(init)-[:REFERENCES]->(final)
```

Created 2866 relationships, completed after 466136 ms.

3. Create *Conference* when raw exists and type equal to 0:

```
call apoc.load.json("test.json") yield value
UNWIND value AS art
WITH art WHERE art.venue.raw IS NOT NULL AND art.venue.type = 0
MATCH(article:Publication{id: art._id})
MERGE(venue:Conference{raw:art.venue.raw}) ON CREATE SET
    venue.name = art.venue.name_d
MERGE(article)-[pub:PUBLISHED]->(venue) ON CREATE SET
    pub.issue = art.venue.issue,
    pub.volume = art.venue.volume,
    pub.issn = art.venue.issn,
    pub.online_issn = art.venue.online_issn,
    pub.publisher = art.venue.publisher
```

Added 2133 labels, created 2133 nodes, set 22676 properties, created 4846 relationships, completed after 22029 ms.

4. Create *Journal* when raw exists and type equal to 1:

```
call apoc.load.json("test.json") yield value
UNWIND value AS art
WITH art WHERE art.venue.raw IS NOT NULL AND art.venue.type = 1
MATCH(article:Publication{id:art._id})
MERGE(venue:Journal{raw:art.venue.raw}) ON CREATE SET
    venue.name = art.venue.name_d
```

```
MERGE(article)-[pub:PUBLISHED]->(venue) ON CREATE SET
    pub.issue = art.venue.issue,
    pub.volume = art.venue.volume,
    pub.issn = art.venue.issn,
    pub.online_issn = art.venue.online_issn,
    pub.publisher = art.venue.publisher
```

Added 256 labels, created 256 nodes, set 1681 properties, created 351 relationships, completed after 6576 ms.

5. Create *Generic Venue* when `raw` doesn't exist or `type` different from 0 or 1:

```
call apoc.load.json("test.json") yield value
UNWIND value AS art
WITH art WHERE art.venue.raw IS NOT NULL AND (art.venue.type IS NULL OR (
   art.venue.type <> 1 and art.venue.type <> 0))
MATCH(article:Publication{id:art._id})
MERGE(venue:GenericVenue{raw:art.venue.raw}) ON CREATE SET
    venue.name = art.venue.name_d
MERGE(article)-[pub:PUBLISHED]->(venue) ON CREATE SET
    pub.issue = art.venue.issue,
    pub.volume = art.venue.volume,
    pub.issn = art.venue.issn,
    pub.online_issn = art.venue.online_issn,
    pub.publisher = art.venue.publisher
```

Added 586 labels, created 586 nodes, set 5122 properties, created 1116 relationships, completed after 8705 ms.

6. Create *Field of Study* and `REGARD` relationship between *FoS* and *Publication*:

```
call apoc.load.json("test.json") yield value
UNWIND value AS pub
UNWIND pub.fos AS f
MATCH(publication:Publication{id:pub._id})
MERGE(fos:Fos{name:f})
MERGE (publication)-[:REGARDS]->(fos)
```

Added 13210 labels, created 13210 nodes, set 13210 properties, created 59740 relationships, completed after 371223 ms.

# 6 | Commands and Queries

## 6.1. Commands

We have identified the following `INSERT` and `UPDATE` commands in order to show the system basic functionalities.

### 6.1.1. Insert two authors in the system

Assuming they are not present in the dataset, we simply used the `CREATE` to create those instances.

```
CREATE (author1:Author {id: "54857748dabfae8a11fb2a1e", name: "Emanuele Della
    Valle", orcid: "0000-0002-5176-5885"})
CREATE (author2:Author {id: "53f487fbdabfaee4dc8b1e68", name: "Alessio Bernardo
    ", orcid: "0000-0002-3492-0345"})
```

### 6.1.2. Insert Journal Venue where the paper has been published

Assuming they are not present in the dataset, we simply used the `CREATE` to create an instance of *Venue (Journal)* specifying where it was published.

```
CREATE (journal:Journal {raw: "ESA", name: "Expert Systems with Applications"})
```

### 6.1.3. Create (if do not exist) publication's fields of study

We simply used the `MERGE` to create three instances of *Field Of Study*

```
MERGE (fos1:Fos{name:"Computer Science"})
MERGE (fos2:Fos{name:"Stream Reasoning"})
MERGE (fos3:Fos{name:"Big Data"})
```

### 6.1.4. Create a publication and his relationships with already existing entities

At the beginning we MATCH two specific *Authors*, three specific *Fields of Study* and one specific *Journal*. Then, we create the *Publication* and the relationships with the entities listed before

```
MATCH (author1:Author),(author2:Author) WHERE author1.id = "54857748
    dabfae8a11fb2a1e" AND author2.id = "53f487fbdabfaee4dc8b1e68"
MATCH (fos1:Fos),(fos2:Fos),(fos3:Fos) WHERE fos1.name="Computer Science" AND
    fos2.name="Stream Reasoning" AND fos3.name="Big Data"
MATCH (journal:Journal) WHERE journal.raw = "ESA"
CREATE (pub:Publication{id: "53e99f86b7612d9702859fdf",
    doi: "10.1016/j.eswa.2022.116630",
    title: "An extensive study of C-SMOTE, a Continuous Synthetic Minority
    Oversampling Technique for Evolving Data Streams",
    year: 2022,
    n_citation: 3,
    keywords: ["Evolving Data Stream","Streaming","Concept drift","Balancing"],
    abstract: "Streaming Machine Learning (SML) studies algorithms that update
    their models, given an unbounded and often non-stationary flow of data
    performing a single pass.",
    lang: "en"})

//Creation of relationship WRITES between authors and the article
CREATE (author1)-[:WRITES{org: "Politecnico di Milano",
    orgid: "5b86c975e1cd8e14a3d351a3"}]->(pub),
    (author2)-[:WRITES{org: "Politecnico di Milano",
    orgid: "5b86c975e1cd8e14a3d351a3"}]->(pub)

//Creation of relationship REGARDS between article and fields of study
CREATE (pub)-[:REGARDS]->(fos1),
    (pub)-[:REGARDS]->(fos2),
    (pub)-[:REGARDS]->(fos3)

//Creation of relationship PUBLISHED between the article and the journal
//Note that in this case fields online_issn and issue are not available
CREATE (pub)-[published:PUBLISHED{volume:"196",
    issn: "0957-4174",
    publisher:"Elsevier"}]->(journal)
```

### 6.1.5. Match the publications referenced by the article created before

We MATCH the two publications referred by our article and we create REFERENCES relationships that link the publications already present in the dataset with the article just created

```
MATCH (pub1:Pubblication),(pub2:Pubblication)
WHERE pub1.id = "53e99fe4b7602d97028bf743" AND pub2.id="53
   e99fddb7602d97028bc085"
CREATE (pub)-[:REFERENCES]->(pub1), (pub)-[:REFERENCES]->(pub2)
```

### 6.1.6. Increment publication's n_citation attribute by 1

It's an update command that increments the number of citations of the matched *Publication*

```
MATCH (article:Publication)
WHERE article.id = "53e99f86b7612d9702859fdf"
SET article.n_citation = article.n_citation + 1
```

## 6.2. Queries

We have identified the following queries in order to show the system basic functionalities.

### 6.2.1. Query 1 - 3 nodes, conditions, aggregations, limits

This query return all the venues where have been published the articles of the author that has written the max number of articles

**Description:** for each author we count how many articles he wrote, then order by the count of written articles by descending order and keep only the top 1 author. Then we match all the articles written by this author and then we return the raw of the venue where these articles were published on

```
MATCH(author:Author)-[:WRITES]->(article:Publication)
WITH author, count(*) AS articleWritten
ORDER BY articleWritten DESC
WITH author AS authorMax, articleWritten LIMIT 1
MATCH(authorMax:Author)-[:WRITES]->(article)
MATCH(article:Publication)-[:PUBLISHED]->(venue)
RETURN venue.raw
```

### 6.2.2. Query 2 - 4 (+1) nodes, conditions, aggregations, limits

Starting from the author found in query 1 we want to find the article with the max number of co-author and return the venue and fields of study

**Description:** we match a specific author given its id, then we match all his articles and then all the co-authors of his articles. Then, for each article we count the number of co-authors and order them by descending order wrt the number of co-authors. We keep the top article, then we match its venue and its fields of study and return them by collecting the fos into a list

```
MATCH(author:Author{id:'548d281cdabfae8a11fb4ea1'})
MATCH(author)-[:WRITES]->(article)
MATCH(article)<-[:WRITES]-(coAuth)
WITH article , count(*) AS nCount
ORDER BY nCount DESC LIMIT 1
MATCH(article)-[:PUBLISHED]->(venues)
MATCH(article)-[:REGARDS]->(fos)
RETURN venues.raw AS Venue, collect(fos.name) AS FieldsOfStudy
```

### 6.2.3. Query 3 - 3 nodes, conditions

This query finds all authors that have worked together more than once (on the same field of study)

**Description:** we match 2 different authors and 2 different articles in which they collaborated, matching also all their fields of study, then we filter by keeping only those who have at least one field of study in common. Then we return the name of the two authors

**Note:** the condition on the authors id needed to ensure that every couple is returned only once

**Note:** query result picture is partial

```
MATCH(aut1:Author)-[:WRITES]->(art1)-[:REGARDS]->(fos1)
MATCH(aut1:Author)-[:WRITES]->(art2)-[:REGARDS]->(fos2)
MATCH(aut2:Author)-[:WRITES]->(art1)
MATCH(aut2:Author)-[:WRITES]->(art2)
WHERE (art1)<-[:WRITES]-(aut2) AND
    (art2)<-[:WRITES]-(aut2) AND
    art1.id <> art2.id AND aut1.id > aut2.id
    AND fos1 = fos2
RETURN DISTINCT aut2.name, aut1.name
```

### 6.2.4. Query 4 - Function (minimum path)

Find shortest path of WRITES links between an author that wrote a publication for a Stanford University and another author that wrote and article for a California University
**Description:** we match 2 authors such that one has published at least once in affiliation with Stanford university and the other with California university. We check that the 2 authors are not the same and then we compute the shortest path between them (composed of WRITES relationships) bounded to 5 steps. Then for all the obtained paths we keep the ones with length > 2 to avoid trivial path
**Note:** graph image is a subset of actual query result (PATH: Kunnle ->...-> Krste)

```
MATCH (auth1:Author), (auth2:Author)
WHERE EXISTS {
    MATCH (auth1)-[w:WRITES]->()
    WHERE w.org CONTAINS 'Stanford'}
AND EXISTS {
    MATCH (auth2)-[v:WRITES]->()
    WHERE v.org CONTAINS 'California'}
AND auth1 <> auth2
MATCH p = shortestPath((auth1)-[:WRITES*1..5]-(auth2))
WHERE length(p) > 2
RETURN p, auth1.name, auth2.name
```

### 6.2.5. Query 5 - 3 nodes, conditions, aggregations, limits

Find the venue with the highest average number of citations of its publications starting from year 1990 and return the most frequent field of study of its publications
**Description:** we match all the publications that have the year attribute > 1990, then we group with respect to the venue they've been published. Then for each venue we compute the average of the number of citations of the articles that were published on it. After that we order the averages by descending order and keep the greatest one. Then we match all the fields of study of all the articles (ignoring the publication date) that have been published on that venue. After that, we count those fields of study, order by descending order and keep only the most frequent one. Finally we return the venue, its most frequent field of study and the number of occurrences of that field of study

```
MATCH (v)<-[p:PUBLISHED]-(a:Publication)
WHERE a.year > 1990 AND a.n_citation IS NOT NULL
WITH v, avg(a.n_citation) AS mean, count(a) AS n_article
ORDER BY mean DESC
```

```
WHERE n_article > 5
WITH v AS venue, mean, n_article LIMIT 1
MATCH (fos)<-[:REGARDS]-(art)-[:PUBLISHED]->(venue)
WITH fos, count(*) AS fosN, venue
ORDER BY fosN DESC LIMIT 1
RETURN venue.raw AS VenueRaw, fos.name AS FieldOfStudy, fosN AS
    FieldOfStudyOccurrence
```

# 7 | Conclusion

Some interesting conclusions can be drawn from the development of this project: designing a Graph DB allows us to open a new perspective in database creation. This technology enables an efficient visualization of the design choices and it is more flexible than classical relational databases.
Another very useful aspect of this project were the dataset pre-processing operation and the profiling that allows us to deal with a real world problem of managing data.

# A | Appendix A

If you need to include an appendix to support the research in your thesis, you can place it at the end of the manuscript. An appendix contains supplementary material (figures, tables, data, codes, mathematical proofs, surveys, ...) which supplement the main results contained in the previous chapters.

# List of Figures

# List of Tables